towards message based audio systems

Winfried Ritsch
Institute of Electronic Music and Acoustics Graz
Inffeldgasse 10
8010 Graz,
Austria,
ritsch@iem.at

Abstract

The deployment of distributed audio systems in the context of computermusic and audio installation is explored in the paper, expanding the vision of static streaming audio networks to flexible dynamic audio networks. Audiodata is send on demand only. Sharing sources and sinks allows us arbitrary audio networks.

This lead to the idea of message based audio systems, which has been investigated within two use cases: Playing on an Ambisonics spatial audio system, and within a computermusic ensemble.

In a first implementation Open Sound Control (OSC) is used as the content format proposing a definition of Audio over OSC (*AOO*).

Keywords

audio-interfaces, networked audio, OSC, computermusic ensembles, sound installation

1 Introduction

The first idea of a message based audio system came up with the requirement of playing a multi-speaker environment of distributed networked embedded devices from several computers, avoiding a central mixing desk.

Another demand for a message based audio network came up during the development of a flexible audio network within the *ICE*-ensemble[1][IEM, 2011]. A variable number of computermusic musicians sending time bounded audio material with their computers to other participants (for monitoring or collecting audio material), would have caused a complex audio-matrix setup of quasi-permanent network connections with all the negotiations and initializations for these streams. Not only because of the limited rehearsal time, this seems to be both too error prone and an overkill in terms of network load.

The structure of a functional audio-network for ICE, especially during improvising sessions, cannot always be foreseen and is therefore hard to implement as a static network. It is therefore important to be able to easily change the audio network during performance, as musicians come and leave (and reboot). On the other hand, the need for low latency, responsiveness and sufficient audio quality has to be respected even during the dynamic change of network connections. No strict requirements on sample-rates, sample-accurate synchronization and the use of unique audio formats should be made in such situations. It should be possible to freely add or remove audio related devices to/from the system without having to go through complicated setup of audio streams and without having to negotiate meta data between the participants. This should simplify the implementation of the particular nodes.

Of course, special care has to be taken when playing together in an ensemble. Factors like network overload, especially peaks, can lead to bad sound and feedbacks. On the other hand, we also find such situations when playing together in the analog world. In any case, the limits have to be explored during rehearsals.

Setting up continuous streams where audio data, including silence, is send continuously to all possible destinations is an overhead, that can easily touch the limits of available network bandwidth. But also can cause wasteful/costly implementations. If we can send audio from different sources to sinks (like speaker systems) only on demand, simplifies the setup. Also, reducing the needs for negotiation for establishing connections simplifies this task, and therefore stabilizes the setup.

The use of messages for the delivery of audio-signals in a network seems to contradict the usual implementation of real-time audio-processing implementations in digital audio workstations, where mostly continuous synchronized audio streams are used. If these audio messages are send repeatedly in such a way that

[1]IEM (Institute of Electronic Music and Acoustics) Computermusic Ensemble

they can be combined together in time, they can been seen as limited audio data streams and supersede continuous audio streams.

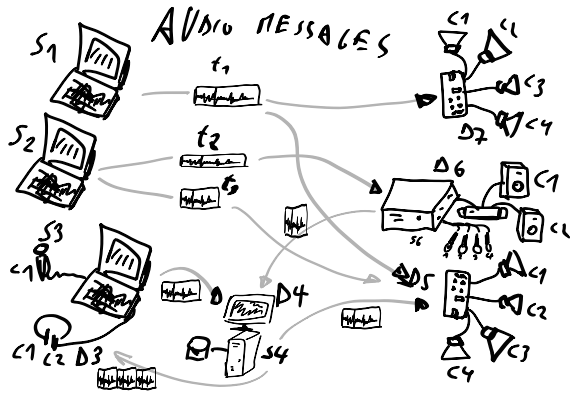

Figure 1: first idea of a message audio system with sources S_n and drains D_n

Summing up these demands, the overall vision is to implement a distributed audio network, where a variable amount of nodes act as sound sources and sound sinks (drains). It should be possible to send audio messages from any source to any sink, from multiple sources simultaneously to a single sink, respectively broadcasting audio messages from one source to multiple sinks. Accordingly, the cross-linking between the audio components is arbitrary, as shown in figure1.

There should not be a "Before you stream audio, you first have to negotiate and connect with ...", Instead, any participant should be able to just send their audio data to others when needed. The receivers should be able to decide how to handle the audio, depending if they can or want to use them.

Following features can be outlined:

- audio signal intercommunication between distributed audio systems
- arbitrary ad hoc connections
- various audio formats, sample-rates
- synchronization and lowest latency possible
- audio-data on demand only

The most common way of communication within local networks is Ethernet. Therefore "Audio over Ethernet" has become a widely used technique. However, there is roughly only a single approach: Stream based audio transmission, representing the data as a continuous sequence. For audio messages as on-demand packet based streams[2] we found no usable implementation (2009). This lead to the design and implementation of a new audio transmission protocol for the demands shown before. As a first approach, an implementation in user space (on the application layer) without the need of special OS-drivers was intended. This can also be seen as the idea of "dynamic audio networks".

2 Audio over OSC

Looking for a modern, commonly used transmission format for messaging systems within the computermusic domain, we found "Open Sound Control" (OSC) [Wright, 2002]. With its flexible address pattern in URL-style and its implementation of high resolution time tags, OSC provides everything needed as a communication format[Schmeder et al., 2010]. OSC specifications points out that it does not require specific underlying transport protocol, but often uses Ethernet network. In our case this would be UDP in a first implementation but is not limited to these. TCP/IP as transport protocol can also be used, but would make some features obsolete and some more complicated, like the requirement for negotiations to initialize connections. Wolfgang Jäger implemented "Audio over OSC" (*AoO*) within a first project at the IEM [Jaeger and Ritsch, 2009]. This was used in tests and "AUON" (all under one net), a concert installation for network art[3]

2.1 the AoO-protocol

The definition of AoO protocol was made with simplicity in mind, targeting also small devices like microcontrollers:

```
AoO message := "#bundle" timestamp
  <format> <channel> [<channel>,...]

format := "/AOO/drain/<d>/format"
  samplerate blocksize overlap mime-type
  [time correction]

channel := "/AOO/drain/<d>/channel/<c>"
  id sequence resolution resampling <data>

d ... number of drain (integer)
c ... channel number (integer)
data ... audio data (blob)
```

[2] not to be mistaken with "streaming on demand" or UDP packets

[3] performed 17.1.2010 in Medienkunstlabor Kunsthaus Graz see `http://medienkunstlabor.at/projects/blender/ArtsBirthday17012010/index.html`

A AoO message is represented by an OSC-bundle with the obligate timestamp. It contains one format message at the beginning and one or more channel messages.

For the addressing scheme the structure of the resources in network is used as the base. Each device in the network with an unique network-address (IP-number and Port number) can have one or more drains. Each of these drains can have one or more channels. There can be an arbitrary amount of drains, and each drain could have an arbitrary amount of channels. An example address of a channel in an device looks like $/AOO/drain/2/channel/3$, where the third channel of the second drain in the device is targeted. $/AOO$ is the protocol specific prefix.

Like described in "Best Practices for Open Sound Control"[Schmeder et al., 2010], REST (Representational State Transfer) style is used. With its stateless representation each message is a singleton containing all information needed.

In OSC, there is a type of query operators called address pattern matching. These can be used to address multiple channels or drains in one message. Since pattern matching can be computational intensive, we propose only to use the "*" wild-char for addressing all channels of a drain or all drains of a device. For instance the channel message $/AOO/drain/2/channel/*$ will use the audio data for all channels of the second drain.

A OSC format message, with for example $/AOO/drain/2/format$ as address string, is always the first item in the bundle and specifies the samplerate, the blocksize and overlap factor as integer, followed by a string as the mime-type of the audio data. The optional time correction factor will be explained at section 2.3.

Integer was chosen in favor for processors without hardware floating point support. Channel specific data information like the id number of the message stream, the sequence number in the channel message allow more easily to detect lost packages. The resolution of a sample and an individual resampling factor is contained in the channel messages, where the resampling factor enables channels to differ from the samplerate specified in the format message, allowing lower rates for sub channels, control streams or higher rates for specific other needs. This also becomes handy if FFT-frames are transmitted as data.

Some of the header data is shown in the following summary example to explain some specific features of the audio transmission:

sampling rate Different sampling rates of sources are possible, which will be resampled in the drain.

blocksize The amount of samples in each package of audio data, which must be greater or equal 1, limited by packet size.

overlapping factor The overlapping factor is 1 (one) by default. Higher values can be used to implement redundancy, to deal with lost packets or needed when sending FFT-frames (in future implementations).

resampling factor is linked to the samplingrate in order to be able to choose the precision of each channel individually using oversampling or similar.

coding of the audio data using the *Multipurpose Internet Mail Extensions* (MIME) standard[Authority, 2009]. In our uncompressed format, the MIME type would be "audio/pcm", whereas "audio/CELP" classifies CELP encoded data.

In order to send usable data, sources have to be aware of the formats a given drain can handle. [4]

data types preferred are uncompressed packets with data types defined by OSC, like 32-Bit float. However, it's also possible to use blobs with an arbitrary bit-length audio data. This can become handy if bandwidth matters. Sources must be aware, which formats can be handled by the drains.

To provide low latency, time-bounded audio transmissions should be sliced into shorter messages and send individually to be reconstructed at the receiver.

2.2 drains

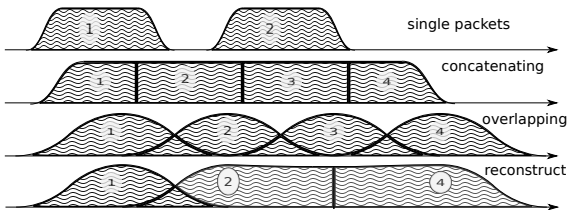

Figure 2: audio messages are arranged as single, combined or overlapped

Sending audio data is simply slicing and adding timestamps. On the other side, receiving

[4]This subject is currently under discussion, and may get changed in the future

means that audio packets have to be rescheduled and synchronized on the time-line, using the timestamp, sequence and id received, and mixed together. Mixing is required either if audio packets come from different sources, have different ids or if they are overlapping (using an overlapping factor greater than one). Audio messages also have to be re-sampled before they are added, to handle with sources with different samplerates. Even if nodes are using the same nominal sample-rate, they are usually not sample-synchronized, since the sample-clocks can drift in time. The re-sampling factor can therefore change dynamically.

For re-arranging the audio packages there is a need to do some sort of labeling of the messages, since it is not clear if they are intended to overlap or are different material. This can be handled via the "identification number" (id). Identical identification numbers means to recognize the material as one material and they can be cross-faded. So these numbers has to has to unique at least at the drain.

The first audio packet has to be faded in and the last faded out. A sequence of audio messages must be concatenated. At least one message has to be buffered to know if a next one arrives. If messages are in overlapping mode, they always have to be cross-faded. If one packet is lost in the overlapping mode, the signal can be reconstructed.

2.2.1 addressing problems

Like described above, to deliver audio messages to a drain, additionally to the drain number and channel number, the address of the device has to be known. A decision was made, that the address is not part of the message, since the sender has to know about the drain on the receiver and the network system has to handle the addressing. Since automatic IP assignment can be used, this could make the argument to simplify the network obsolete, since we devices have no static address.

Like stated in in the vision, we do want negotiations and requests, but in situations where IPs are unknown, we needed a mechanism to grasp it. One implementation was announcement message broadcasted by each drain, with the address and a human readable meaningful name. Even more polite we implemented them as invitation messages, which also states: "ready to receive". This was done every 10 seconds, to limit load and also serves as a live message.

A second problem arose, since broadcasting to all drains with the same number, the destination information is not contained in the audio message, we cannot use broadcast to reduce network load and address specific destinations. For this the drain has to know about the sources it will accept. Anyway this worked fine, but made some additional efforts in communication before.

Anyway addressing is in heavy discussion, has to be tested further on use cases and will probably change in future.

2.2.2 mixing modes

In this first implementation we used two different modes: Mode 1 provides the possibility of summation of the received audio signals and Mode 2 should perform an arithmetic averaging of parallel signals. The reason for this is that summing audio signals with maximum amplitudes each causes distortion. Using Mode 2 this cannot happen. If many sources play into one drain, this can also be seen as a kind of mix to reduce the impact of a single one. Sometimes automatic level control or limiting in the digital domain after adding the signals is the better way to prevent clipping.

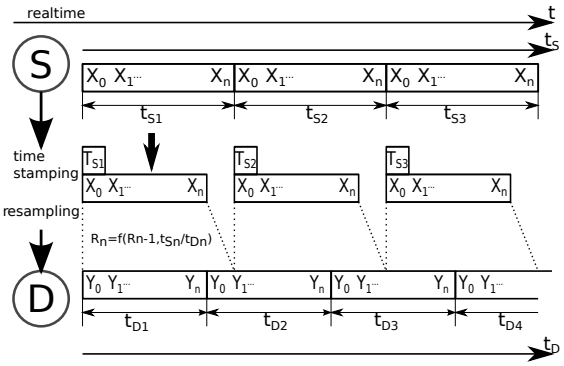

Figure 3: re-sampling rate R_n between source S and drain D is not constant

2.3 timing and sample-rates

Timing is critical in audio-systems, not only for synchronizing audio, but also to prevent jitter noise. Time tags of the packets are represented by a 64 bit fixed point number, as specified by OSC, to a precision of about 230 picoseconds. This conforms to the representation used by the Network Time Protocol (NTP)[Mills et al., 2010].

In fixed buffering mode, the buffer size has to be chosen large enough to prevent dropouts. In the automatic buffer control mode, the drain

should use the shortest possible size for buffering. If packets arrive too late, buffering should be dynamically extended and then slowly reduced.

Since audio packets can arrive with different sample-rates, re-sampling is executed before the audio data is added to the internal sound stream synchronized with the local audio environment. This provides the opportunity to synchronize audio content respecting the timing differences and time drifts between sources and drains. This strategy of resampling is shown in figure 3:

Looking at synchronization in digital audio system, mostly a common master-clock is used for all devices. Since each device has its own audio environment, which may not support external synchronization sources, the time T_Sn of the local audio environment is used to calculate the timestamp for outgoing audio messages.

Using the incoming timestamps from the remote source, we can compare them with the local time t_Dn and correct the re-sampling factor R_n dynamically for each message. The change of the correction should be small if averaged over a longer time, but can be bad for first audio messages received.

For a better synchronization of audio data, a Time Transfer protocol can be used in parallel to synchronize the drain with the source, as a sort of master-clock.

Therefore, as proposed in the OSC specifications, NTP can be used for each node. Another time protocol for synchronization of audio data is the Precision Time Protocol (PTP)[on Sensor Technology, 1588–2002], e.g. also used in AVB[5], allows a more lightweight implementation in local networks and can guarantee a quasi sample-accurate synchronization. PTP is the preferred time protocol to be used with AoO. For these protocols we need a master (or grandmaster) in the network. This is done differently depending on the used implementation of the time protocol.

Since the local time source of a device can differ from the timing of the audio environment, each device needs a correction factor between this time source and the audio hardware time including the time master device. This factor has to be communicated between the devices, so the re-sampling correction factor can be calculated before the first audio message is sent, guaranteeing a quasi sample-synchronous network-wide system starting with the first message send.

2.4 Requests

Asking won't hurt. If the drain provides information about its capabilities, it can be used to optimize and ensure the transmission. However, this information is optional, allowing simple implementations on some nodes, like microcontrollers, that may be unable to accomplish this task. Until now there is no proposal how to implement such requests, instead we used announcement/invitation messages for grasping the sources in the local net.

2.5 Implementation

As a first proof of concept, *AoO* was implemented within user space using Pure Data[Puckette, 1996]. This implementation has shown various problems to be solved in future. Using the network library iemnet[6] additional "externals" have been written in C to extend the OSC-Protocol, split continuous audio signals into packets and mix OSC audio messages in drains. As repository for the GPL open source can be found at the "Opensource@IEM" sourceforge as git repository site at:

`http://sourceforge.net/p/iem/aoo/`

As a first test environment, a number of different open-source audio hardware implementations, using Debian Linux OS-System, has been used. The test patches were written with Pd version 0.42.5, where a central component has been the OSC library of Martin Peach. Later, an implementation for a microcontroller board "escher2"[Algorythmics, 2012] has been created, which has been superseded by small embedded arm-devices, like beaglebones[Foundation, 2013], also using a Debian OS system.

3 message based Ambisonics spatial audio systems

As a first goal, the geodesic sound-dome in Pischelsdorf (with a diameter of 20 m and a height of about 10 m) as an environmental landscape sculpture in Pischelsdorf should transmute into 3D a sound-sphere. Therefore as special hardware and software, a low power solar power driven multichannel Ambisonics system was developed and installed prototypically. This should result in a low cost implementation of multichannel audio system Up to 48 speakers

[5] Audio Video Briding, Standard IEEE 802.1

[6] iemnet project site is `http://puredata.info/downloads/iemnet`

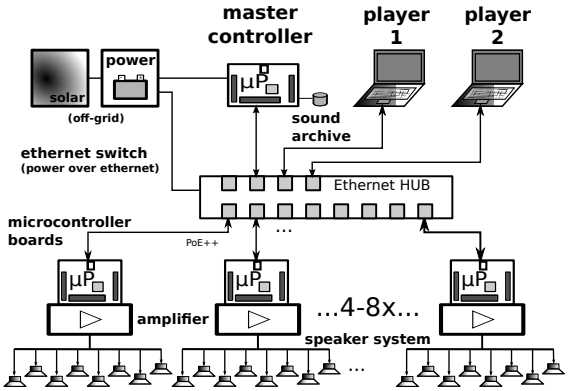

Figure 4: *AoO* with embedded devices for spatial audio system

should be mounted in a hemisphere, forming an Ambisonics sound system. Using 6 nodes, each with 8 speakers, special embedded controllers are used to render the audio in the system (figure 4).

Figure 5: One node with one speaker in the dome

Each node is a small embedded computer equipped with an 8-channel sound-card, including amplifiers and speakers. Each speaker can been calibrated and fed individually. However, since each unit is aware of its speaker positions, it can also render the audio with an internal Ambisonics encoder/decoder combination.

So instead of sending 48 channels of audio to spatialize one or more sources, the sources can be broadcast combined with OSC-spatialization data and the drains render them independently. Another possibility is to broadcast an encoded Ambisonics-encoded multichannel signal, where the devices decode the Ambisonics signal for their subset of speakers. The Sound Environment can be sent from one master controller or any other connected computer.

The first implementation of the nodes has been done with special micro-controller boards *escher2*[Algorythmics, 2012] which drive the custom designed DA-Amp boards. Since these devices have very limited memory (max. 16 samples of 64 channels), standard Linux audio system cannot provide the packets small and fast enough for a stable performance without special efforts, like own driver in kernel space for the packet delivery. Therefore a major problem has been the synchronization and the reliability of the transmission, but providing latency.

One other solution could be, to secure resources like bandwidth and computing time with restricting audio data to be sent on defined time slots: only and one time-slot for each device. Most of the available network-components are able to handle the IP-protocol or even OSC but unfortunately there is no commonly used computer OS, which is able to deliver audio data in dedicated time-slots. Therefore as one implementation of hard real-time networking for real-time Linux, the RTnet[Team, 2002] has been found. It needs a hard-realtime kernel. In a further thought the OSC-Transmission has to be implemented as a Linux-device, coupling with the RT-Net Ethernet driver.

Since 2012 small embedded Linux-systems like the beaglebone black[Foundation, 2013] are available and can be used to drive the DAs with amplifiers. This has been tested recently with good success on a beaglebone black: An acceptable latency of 5-10 ms with 8 out-channels has been achieved .

Figure 6: sounddome as hemisphere, 20 m diameter in cornfield

The main advantage, besides the low cost and autonomous system, is that one or more sound technicians or computer musicians can enter the

dome, plug into the network with their portable devices and play the sound dome either addressing speakers individually, with audio material spatializing live with additional OSC messages or a generated or prerecorded Ambisonics audio material.

3.1 Playing together

Figure 7: first concert of IEM computermusic ensemble ICE playing over a HUB

When specifying an audio-network for playing togehter within an ensemble, a focus was set on the collaborating efforts to be done to gain the unity of the individuals.

So, like a musicians with acoustic instrument, joining a band with Linux audio-computer implies a need for a place where the musician has a "virtual sound space" they can join. So they provide sound sources and need to plugin audio channels on a virtual mixing desk. With *AoO* the participant just needs to connect to the network, wireless or wired, choosing the drains to play to and send phrases of audio with AoO when needed.

For the ICE ensemble Ambisonics as an virtual audio environment was chosen, which can be rendered to different concert halls. Within the Ambisonics each musician can always use the same playing parameters for spatializing her or his musical contribution. So the imagination of the musician is "playing in a virtual 3D environment", sending their audio signals together with 3D-spatial data to a distributed mixing system which is rendering it on the speakers.

Additional there is an audio communication between the musicians, where each musicians can hear into the signal produced by the other, if there is one or on special offered drains send audio intervention to the others for e.g. monitoring purposes. The musicians can do their own monitor mix, depending on the piece and space where the play.

Using a message audio system, each musicians only sends sound data if playing, like audio bursts just notes, or just sending their audio-data to another musicians, who will process this further and so on. There should be no border on the imagination of these situations, (as long it can be grasped by the participants).

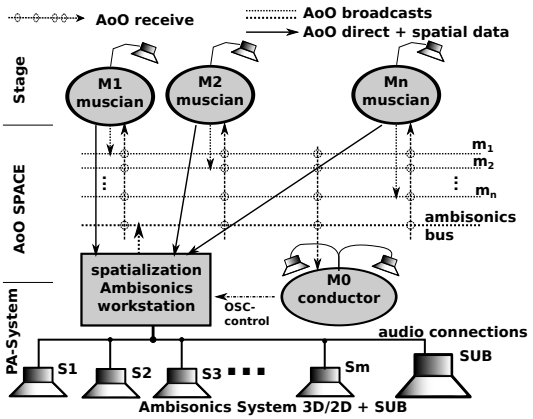

Figure 8: ICE using AoO as space for playing together and on a PA system6

4 state of the work

The *AoO* has been implemented for proof of concept and special applications in a first draft version. The next version should fixate the protocol, after having discussed it in public, in a way that makes it compatible with future protocol upgrades.

The usage of *AoO* in an ensemble has been explored in a workshop with students at the IEM, but the implemented software was not stable enough on the different platforms used for stage performance. This was especially true, when we tried to reach the short latencies needed for concerts. Some more programming efforts has to be done, to guarantee better timing using different computer types, within different Linux-implementations and setups.

Running *AoO* on embedded Linux devices has shown to be successful, if the devices are tweaked for real-time audio usage. The development on the *escher2* (dsPIC33E-)microcontroller board has been abandoned in favor of the new generation of small low power embedded devices with arm processors. A first version of implementation (V1.0) of *AoO* is scheduled for April 2014 for a public installation in the sound-dome, where the Ambisonics audio-system should be finalized for permanent perfor-

mance and open access. More documentation and source code should be released and open-hardware as *AoO*-audio devices should be available.

Special focus is done on using embedded devices with *AoO* as networked multichannel audio hardware interfaces for low cost solutions adding audio processing for calibration filters, beam-forming,... for speaker-systems optional powered over Ethernet.

5 Conclusions

Starting as a vision, these experiments and implementations have shown, that message based audio systems can enhance the collaboration in ensembles, playing open audio systems. Also network art projects using the Internet can use *AoO* to contribute to sound installation from outside, just knowing the IP and ports to use.

The implementation is far from being complete, and more restrictions will be included in order to simplify the system. Synchronization and re-sampling is not perfect, but usable for most cases and it has been shown, that audio message systems can work reliable in different situations.

Audio message systems can also be implemented in other formats than OSC and lower layers of the Linux OS, like jack-plugins or ALSA-modules as converters between message based audio system and synchronous data flow models.

For really low latency (below 1 ms) using *AoO* as audio over Ethernet system, kernel-drivers must be developed and with time-slotted Ethernet transmissions, systems with latencies down to 8 us on transmission time can be implemented using hard RT-systems.

6 Acknowledgements

Thanks to ... my colleagues on the IEM supporting me with their help, especially Wolfgang Jäger for a first implementation as a sound-engineering project. Also for helping set up the "Klangdom" especially to Marian Weger, Matthias Kronlachner and the cultural initiative K.U.L.M. in Pischelsdorf and the members of the ICE Ensemble helping to experiment and many others. Thanks also for corrections of this paper and useful hints, to enhance the understanding.

References

Atelier Algorythmics. 2012. escher development. `http://algo.mur.at/projects/microcontroller/escher`. [Online; accessed 1-Feb-2014].

Internet Assigned Numbers Authority. 2009. MIME Media Types. `http://www.iana.org/assignments/media-types/`. [Online; accessed 1-Feb-2014].

The BeagleBoard.org Foundation. 2013. beaglebone black. `http://beagleboard.org/products/beaglebone\%20black`. [Online; accessed 1-Feb-2014].

Winfried Ritsch IEM. 2011. IEM Computer Music Ensemble. `http://iaem.at/projekte/ice`. [Online; accessed 1-Feb-2014].

Wolgang Jaeger and Winfried Ritsch. 2009. AOO. `http://iem.kug.ac.at/en/projects/workspace/2009/audio-over-internet-using-osc.html`. [Online; accessed 12-Dez-2011].

D. Mills, J. Martin, J. Burbank, and W. Kasch. 2010. Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905 (Proposed Standard), June.

Technical Committee on Sensor Technology. 1588–2002. *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. The Institute of Electrical and Electronics Engineers, Inc. (Hrsg.), New York, ieee std. edition, November.

M. Puckette. 1996. Pure Data. In *Proceedings, International Computer Music Conference.*, page 224–227, San Francisco.

Andrew Schmeder, Adrian Freed, and David Wessel. 2010. Best Practices for Open Sound Control. In *Linux Audio Conference*, Utrecht, NL, 01/05/2010.

RTnet Development Team. 2002. RTNET. `http://rtnet.org/`. [Online; accessed 1-Feb-2014].

Matt Wright. 2002. The open sound control 1.0 specification. `http://opensoundcontrol.org/spec-1\_0`. [Online; accessed 1-Feb-2014].